**Lab 4 – PaletteSet and Database Events**

In this lab, we will add a PaletteSet and add database events. In the events we will add data about entities added to the drawing to a treeview control in a paletteSet.

Open your Lab3 project and add a UserControl. Go to the Project menu and select “Add User Control”. (Or open the Lab4 project that already has the UserForm and the steps for lab).



Accept the default name (UserControl1.vb)

Display the Toolbox (Ctrl+Alt+X) and add a TreeView control to the UserControl.



Anchor all of the sides of the TreeView control using the properties window as seen in this screenshot:



Change the protection property of the treeView control to Public:



Copy the steps below to your project. Place the steps after the closing curly brace of the “addAnEnt” function from Lab3. Work through the steps to create the PaletteSet and the Database Events.

// 1. Add a Reference to PresentationCore. (Use the .NET tab on

// the Add Reference dialog. This is needed for the PaletteSet

// we will declare in step 3.

// 2. Use the using Statement for namespace Autodesk.AutoCAD.Windows

// 3. Declare a PaletteSet variable (global) as a PaletteSet. (It will

// only be created once). Add this declaration after AddAnEnt function

// from Lab 3.

// 4. Declare a variable as UserControl1. This is the control created

// in the steps above. This control

// will be housed by the PaletteSet declared in step 3. Need to use the

// NameSpace of the UserControl1 in the declaration. (Lab4)

// 5. Add an new command named palette. Use the CommandMethod

// attribute and create the function that will run when the command

// is run in AutoCAD.

// Note: Put the closing curly brace after step 10.

// 6. Add an "if" statement and check to see if the

// PaletteSet declared in step 3 is equal to null. It will be

// null the first time the command is run.

// Note: Put the closing curly brace after step 9

// 7. The PaletteSet is nothing here so we create a a new PaletteSet

// with a unique GUID. Use the new keyword. Make the Name Parameter

// "My Palette". For the ToolID parameter generate a new GUID.

// On the Tools menu select "Create Guid". On the Create GUID

// Dialog select "Registry Format" Select New GUID and the copy.

// Paste the GUID to use as the new System.Guid. Replace the curley

// braces with double quotes. (The parameter for New Guid is a string)

// 8. Instantiate the UserControl1 variable created in

// step 4. Use the new keyword. (New UserControl1 - need to

// use the namespace too)

// This control houses the tree control.

// 9. Add the UserControl to the PaletteSet. Use the Add method

// of the PaletteSet instantiated in step 7. Use "Palette1" for the

// name parameter and the control instantiated in step 8 for the

// second parameter.

// 10. Display the paletteset by making the Visible property of the

// PaletteSet equal to true. The second time the command is run

// this is the only code in this procedure that will be processed.

// 11. Add a command named "addDBEvents. Use the CommandMethod attribute

// and add the function that will run when the commmand is run in AutoCAD

// Note: Put the closing curly brace after step 20

// 12. use an if statement and see if the palette

// created in step 4 Is null.

// Note: put the closing curley brace after step 15

// 13. Declare and intantiate a Editor object. Use the Editor

// property of Application.DocumentManager.MdiActiveDocument

// 14. Use the WriteMessage method of the Editor variable

// created in step 13. Use this for the message parameter

// "\n" + "Please call the 'palette' command first"

// 15. return

// 16. Declare a Database variable and instantiate it by making it

// equal to the Database property of the

// Application.DocumentManager.MdiActiveDocument

// 17. Connect to the ObjectAppended event. Use the ObjectAppended

// event of the database variable created in step 16. Use

// += and use the new statement and create a new ObjectEventHandler. For

// the target parameter use the name of a function we will create in step 21.

// (callback\_ObjectAppended).

// 18. Connect to the ObjectErased event. Use the ObjectErased

// event of the database variable created in step 16. Use

// += and use the new statement and create a new ObjectErasedEventHandler. For

// the target parameter use the name of a function we will create in step 24.

// (callback\_ObjectErased).

// 19. Connect to the ObjectReappended event. Use the ObjectReappended

// event of the database variable created in step 16. Use

// += and use the new statement and create a new ObjectEventHandler. For

// the target parameter use the name of a function we will create in step 32.

// (callback\_ObjectReappended).

// 20. Connect to the ObjectUnappended event. Use the ObjectUnappended

// event of the database variable created in step 16. Use

// += and use the new statement and create a new ObjectEventHandler. For

// the target parameter use the name of a function we will create in step 35.

// (callback\_ObjectUnappended).

// 21. Create a private function named callback\_ObjectAppended. (returns void)

// This is the function that will be called when an Object is Appended to

// the Database. (The name needs to be the name used in the Delegate parameter

// of step 17). The first parameter is an object. (use sender as the name of

// the Object). The second parameter is an ObjectEventArgs.

// (Use e as the name of the ObjectEventArgs)

// Note: Put the closing curly brace after step 23

// 22. Declare a TreeNode variable. (System.Windows.Forms.TreeNode).

// Note: You can save some typing by adding a using statement and add the namespace.

// Instantiate it using the Add method of the Nodes property of the TreeView on the

// UserControl() created in step 4. Use the ObjectEventArgs passed into the function for

// the string parameter and use the "Type" of DBObject. (e.DBObject.GetType().ToString())

// 23. Make the Tag property of the node created in step 22 equal to the ObjectId of

// the appended object. This will allow us to record it's ObjectId for recognition in

// other events. Use e.DBObject.ObjectId.ToString()

// 24. Create a private function named callback\_ObjectErased. (returns void)

// This is the function that will be called when an Object is erased from the

// Database. (The name needs to be the name used in the Delegate parameter of

// step 18). The first parameter is an object. (use sender as the name of the

// Object). The second parameter is an ObjectErasedEventArgs.

// Use e as the name of the ObjectErasedEventArgs)

// Note: Put the closing curly brace before step 32

// 25. use an "if else" statement and check the Erased property of the

// ObjectErasedEventArgs passed into the function. (e.Erased)

// Note: Put the closing curly brace and "else" stament before step 30.

// put the closing curly brace for the "else after step 31

// 26. Here we will search for an object in the treeview control so it can be removed.

// Create a foreach statement. Use node for the element name and the type is

// System.Windows.Forms.Treenode. The group paramater is the Nodes in the TreeView.

// (myPalette.treeView1.Nodes)

// Note: put the closing curly brace below step 29.

// 27. Use an "if" statement. Test to see if the node Tag is the ObjectId

// of the erased Object. Use the DBObject property of the of the

// ObjectErasedEventArgs passed into the event. (e.DBObject.ObjectId.ToString())

// Note: put the closing curly brace above the closing curley brace for the for loop

// 28. Remove the node by calling the Remove method. (The entity was

// erased from the drawing).

// 29. Exit the For loop by adding a break statement.

// 30. If this is processed it means that the object was unerased. (e.Erased was false)

// Declare a System.Windows.Forms.TreeNode use newNode as the name. Instantiate it by

// using the Add method of the Nodes collection of the TreeView created in previous steps.

// Use the Type of the object for the parameter.

// e.DBObject.GetType().ToString()

// 31. Make the Tag property of the node created in step 30 equal to the ObjectId of

// the unerased object. This will allow us to record it's ObjectId for recognition in

// other events. Use e.DBObject.ObjectId.ToString()

// 32. Create a private function named callback\_ObjectReappended. This is the func that

// will be called when an Object is ReAppended to the Database. (The name needs to be

// the name used in the Delegate parameter of step 19). The first parameter is an

// object. (Use sender as the name of the Object). The second parameter is

// an ObjectEventArgs. (use e as the name of the ObjectEventArgs)

// Note: Put the closing curly brace after step 34

// 33. Add the class name of the object to the tree view

// Declare a TreeNode variable. (System.Windows.Forms.TreeNode). Instantiate

// it using the Add method of the Nodes property of the TreeView on the UserForm1

// created in step 4. Use the ObjectEventArgs passed into the method for the string

// parameter and use the "Type" of DBObject. (e.DBObject.GetType().ToString())

// 34. Record its id for recognition later

// Make the Tag property of the node created in step 33 equal to the ObjectId of

// the unerased object. This will allow us to record it's ObjectId for recognition in

// other events. Use e.DBObject.ObjectId.ToString()

// 35. Create a private Sub named callback\_ObjectUnappended. (returns void) This is the

// function that will be called when an Object is UnAppended from the Database.

// (The name needs to be the name used in the Delegate parameter of step 20).

// The first parameter is an object. (Use sender as the name of the Object).

// The second parameter is an ObjectEventArgs.

// (Use e as the name of the ObjectEventArgs)

// Note: Put the closing curly brace after step 39

// 36. Here we will search for an object in the treeview control so it can be removed.

// Create a foreach statement. Use node for the element name and the type is

// System.Windows.Forms.TreeNode. The group paramater is the Nodes in the TreeView.

// (myPalette.treeView1.Nodes)

// Note: Put the closing curly brace after step 39

// 37. Use and "if" statement and see if the node is the one we want.

// compare the node.Tag to the ObjectId. (use e.DBObject.ObjectId.ToString)

// Note: Put the closing curly brace after step 39

// 38. If we got here then this is the node for the unappended object.

// call the Remove method of the node.

// 39. Exit the For loop by adding a break.